② 若(K-1)在最后一列但不在第一行，则将 K 填在第一列，(K-1)所在行的上一行；

③ 若(K-1)在第一行最后一列，则将 K 填在(K-1)的正下方；

④ 若(K-1)既不在第一行，也不在最后一列，如果(K-1)的右上角还未填数，则将 K 填在(K-1)的右上方，否则将 K 填在(K-1)的正下方。

现给定 N，请按上述方法构造 $N\times N$ 的幻方。

```
cin >> n;
a[1][n / 2 + 1] = 1; //将 1 写在第一行正中间
x = 1;
y = n / 2 + 1; // x 和 y 分别代表此时填数的位置
m = n * n;
for (int i = 2; i <= m; i++)
{
    int dx,dy;
    if (x == 1 && y != n)
    { //按①的规则填数
        dx = n;
        dy = y + 1;
    }
    else if (y == n && x != 1)
    { //按②的规则填数
        dy = 1;
        dx = x - 1;
    }
    else if (x == 1 && y == n)
    { //按③的规则填数
        dx = x + 1;
        dy = y;
    }
    else if (x != 1 && y != n)
    { //按④的规则填数
        if (!a[x - 1][y + 1])
        { // 右上方还未填数,填在右上方
            dx = x - 1;
            dy = y + 1;
        }
        else
        { //否则填在正下方
            dx = x + 1;
            dy = y;
        }
    }
    a[dx][dy] = i;
```

```
            x = dx;
            y = dy;
        }
        for (int i = 1; i <= n; i++)
        { //输出结果
            for (int j = 1; j <= n; j++)
                cout << a[i][j] << " ";
            cout << endl;
        }
```

典型题目

1. NOIP2003 普及组 乒乓球
2. NOIP2009 普及组 多项式输出
3. NOIP2010 普及组 接水问题
4. NOIP2011 提高组 铺地毯
5. NOIP2012 普及组 寻宝
6. NOIP2012 提高组 Vigenère 密码
7. NOIP2014 提高组 生活大爆炸版石头剪刀布
8. NOIP2015 提高组 神奇的幻方
9. NOIP2016 普及组 海港
10. NOIP2016 提高组 玩具谜题
11. NOIP2017 提高组 时间复杂度
12. NOIP2018 普及组 龙虎斗
13. CSP2019-J 公交换乘
14. CSP2021-J 网络连接

（李绍鸿　谢秋锋）

1.4.3 基础算法

1.4.3.1 贪心法

贪心法（greedy method）又称贪婪算法，是一种在每次决策时采取当前最优策略的算法，即由局部最优得到整体最优，而任何对于局部最优策略的改变都会使得整体结果变差。通常可以使用反证法、数学归纳法、邻项交换法（exchange argument）等方法来证明贪心法的正确性。

代码示例

如排队接水问题，该问题可以用贪心法来求解。要使得 n 个人的平均等待时间最小，即可以求出一种排列方式，使得所有人的等待总时间最少。等待总时间为每个人的

接水时间与后续等待人数乘积的总和，因此，对接水时间从小到大排序，使得到的等待总时间最少，同时也使得 n 个人的平均等待时间最少。用贪心法实现排队接水问题，主要代码如下。

```
long long sum = 0;
for (int i = 1; i <= n; i++)
{
    printf("%d ",a[i]. num); //按接水时间从小到大排好序后的队伍就是接水队伍
    if (i != n)
        sum += (n - i) * a[i]. ti; //第 i 个人接水时,其后 n-i 个人在等待,总等待时间
                                    为 a[i]. ti*(n-i)
}
printf("\n%0.2f",double(sum * 1.0 / n));
```

参考词条

排序算法

延伸阅读

THOMAS H C，CHARLES E L，RONALD L R，et al. 算法导论(原书第 3 版)[M]. 殷建平，徐云，王刚，等译. 北京：机械工业出版社，2013：237-254.

典型题目

1. NOIP2004 提高组 合并果子
2. NOIP2007 普及组 纪念品分组
3. NOIP2010 普及组 三国游戏
4. NOIP2015 普及组 推销员
5. NOIP2011 提高组 观光公交
6. NOIP2012 提高组 国王游戏
7. NOIP2013 提高组 积木大赛
8. NOI2014 起床困难综合症
9. USACO2006Feb Stall Reservations
10. USACO2007Nov Sunscreen

(李绍鸿　谢秋锋)

1.4.3.2 递推法

递推法(iterative method)指的是从给定条件出发，依据某种递推关系，依次推出所求问题的中间结果及目标结果的方法。其中，给定条件可能是问题本身明确给出的，也有可能是通过对问题的分析与化简后而确定的。

递推计算的顺序，可以与问题中的时序、逻辑顺序等一致，也可以恰好相反。按照

顺序是否一致，递推法可以分为顺推和逆推两种：从初始结果出发，逐步推向最终结果，被称为顺推；从最终结果出发，逐步推向初始结果，被称为逆推。无论顺推还是逆推，递推法的关键都是要准确地确定递推关系。如果递推关系还可以表示成数学公式，则将相应公式称为递推公式。

Fibonacci 数列第 n 项的值是可用递推法求解的典型问题。其中，初始条件为 $F_0=F_1=1$，递推公式为 $F_i=F_{i-1}+F_{i-2}$。

典型题目

1. NOIP2001 普及组 数的计算
2. NOIP2001 提高组 数的划分
3. NOIP2003 普及组 栈

（陈奕哲　谢秋锋）

1.4.3.3 递归法

递归法(recursive method)是把一个大型复杂的问题层层转化为一个与原问题相似的、规模较小的问题来求解的方法。

在问题求解过程中，能够用递归法解决的问题，一般需要满足如下要求：

(1) 需要求解的问题可以化为子问题求解，其子问题的求解方法与原问题相同，只是问题规模缩小；

(2) 递归调用的次数是有限的，必须有递归结束的条件，即递归的边界。

代码示例

递归法解决汉诺塔(Hanoi tower)问题的代码如下。

```
void mov(int n,char a,char c,char b)
{
    if (n == 0)
        return; // 递归的边界,没有盘子可以移动了,结束本层递归
    mov(n - 1,a,b,c); // 先将上面的 n-1 个盘子,从 A 柱移动到 B 柱,可以借助于 C 柱
    movedisc(a,c); // 将 A 柱上剩余的一个盘子,直接移动到 C 柱上
    mov(n - 1,b,c,a); // 将 B 柱上的 n-1 个盘子,从 B 柱上移动到 C 柱上,可以借助于 A 柱
}
```

参考词条

1. 栈
2. 递归函数

延伸阅读

王晓东．计算机算法设计与分析[M]．3 版．北京：电子工业出版社，2007：9-14.

典型题目

1. NOIP2001 提高组 数的划分
2. NOIP2001 普及组 求先序排列
3. NOIP2007 普及组 Hanoi 双塔问题

（李绍鸿　谢秋锋）

1.4.3.4　二分法

二分法(bisection method)是一种在有序序列中查找某一个元素的方法。其基本思路是每次用有序序列的中间元素与待查找元素比较，根据比较结果将区间缩小到左边的一半或右边的一半，经过大约 $\log_2 n$ 次比较即可在长度为 n 的有序序列中确定待查找元素的位置。二分法的基本流程如下。

(1) 定义 L 和 R 分别表示区间的左端点和右端点，初始化为有序序列的最小和最大的下标。

(2) 令 M 为区间的中点，将有序序列中的第 M 个元素与待查找元素比较，根据比较结果确定待查找元素在左边一半还是右边一半，并更新区间。

(3) 重复执行步骤(2)，直到区间中只剩下一个元素或区间为空。如果区间中剩下的一个元素为待查找元素，则找到待查找元素，否则待查找元素不在序列中。

二分法也可用于查找第一个大于等于待查找元素的位置、最后一个小于等于待查找元素的位置等。在编程实现时，中点是选取偏左的还是偏右的以及比较结束后剩余区间是否包含中点需要根据具体问题分析。

对于自变量为整数或实数的函数，如果存在一个自变量的分界点，使得在分界点的一侧条件都满足，另一侧条件都不满足，则二分法可以应用在此函数上用于找到该分界点。

对于最优化问题，如果对于一个期望的答案大小 A 可以定义检测函数判断是否存在等于或优于 A 的答案，则可以在可行的答案范围内找到该检测函数的分界点，即最优答案。该方法称为二分答案法。

代码示例

二分法查找的核心代码如下。

```
// 该写法针对的是整数域上的二分
// 在实数域上二分时,判断条件改为 l+eps<=r,其中 eps 为自设的精度值
// 在实数域上二分时,区间的调整改为 l=mid 和 r=mid
int binary_search(int l, int r, int x)
{
    int mid, res = -1;
    while (l <= r)
```